Soft Computing (2015) manuscript No.
(will be inserted by the editor)

Evolutionary Approaches to Signal Decomposition in an Application Service Management System

Tomasz D. Sikora · George D. Magoulas

Received: date / Accepted: date

Abstract The increased demand for autonomous control in enterprise information systems has generated interest on efficient global search methods for multivariate datasets in order to search for original elements in time–series patterns, and build causal models of systems interactions, utilization dependencies, and performance characteristics. In this context, activity signals deconvolution is a necessary step to achieve effective adaptive control in Application Service Management. The paper investigates the potential of population–based metaheuristic algorithms, particularly variants of particle swarm, genetic algorithms and differential evolution methods, for activity signals deconvolution when the application performance model is unknown a–priori. In our approach, the Application Service Management System is treated as a black– or grey– box, and the activity signals deconvolution is formulated as a search problem, decomposing time–series that outline relations between action signals and utilization–execution time of resources. Experiments are conducted using a queue–based computing system model as a test–bed under different load conditions and search configurations. Special attention was put on high–dimensional scenarios, testing effectiveness for large–scale multivariate data analyses that can obtain a near–optimal signal decomposition solution in a short time. The experimental results reveal benefits, qualities and drawbacks of the various metaheuristic strategies selected for a given signal deconvolution problem, and confirm the potential of evolutionary–type search to effectively explore the search space even in high–dimensional cases. The approach and the algorithms investigated can be useful in support of human administrators, or in enhancing the effectiveness of feature extraction schemes that feed decision blocks of autonomous controllers.

Keywords Application Performance Management, Application Service Management, Autonomous Control, Data Analysis, Data Modeling, Metaheuristics, Multidimensional Deconvolution, Optimization, Signal Extraction, Service Level Agreement.

Tomasz D. Sikora · George D. Magoulas
Department of Computer Science and Information Systems
Birkbeck, University of London
Malet Street, London WC1E 7HX

Tomasz D. Sikora
Phone: +44 20-3287-8256, +48 787-622-787
E-mail: sikora.t@gmail.com

George D. Magoulas
Phone: +44 20-7631-6717
E-mail: gmagoulas@dcs.bbk.ac.uk

1 Introduction

The complex nature of enterprise information systems requires processes and tools for monitoring and managing application services and their performance levels in order to meet end–users requirements. Services form an important type of technology in today's business environment and their availability, performance and usage is defined by service–level agreements that IT organizations should comply with. The field of Application Service Management provides sets of well–defined processes, related technologies and tools that aim to detect poor performance levels, identify the cause of service failures, perform appropriate control actions, and rectify performance degradation in enterprise systems.

Despite advances in this area, the operating environment constantly introduces new challenges, as data and information are delivered to end–users' through various

platforms, combining different types of applications, operating systems, and networks. Moreover, enterprise infrastructure services normally deal with large number of complex business requests, or queries, and volumes of data generated by the organization's business processes. Lastly, new business needs emerge that require constant adaptation of the software infrastructure and as a result changes to runtime characteristics.

Resources, system and services activity monitoring tools provide an outlook of the enterprise system operation but gathering measurements and making inferences from the data remain challenging in Application Performance and Service Management (APM and ASM) fields, where usage and performance metrics are acquired from all significant elements of the enterprise systems and from all tiers of its architecture (Haines, 2006; Sydor, 2010; Grinshpan, 2012). Thus, despite the fact that performance profiling and services monitoring are widely used in the enterprise, SaaS and Cloud based businesses, most of the control actions are taken either by humans, or in a semi–automatic way. Typically, these are, for example, semi–automated routines, which are implemented using scripts or rules defined in ASM, and infrastructure formation tools that allow dynamic resources provisioning based on current and historical allocations but need to be maintained by administrators.

In this context, human decision makers need to react constantly to changing system conditions, or requirements, in order to optimize the performance against a set of objective functions, such as those defined in Service Level Agreements (SLAs), or, more generally, specified by Quality of Service (QoS) related metrics. Indeed, human operators or administrators are able to identify which actions are responsible for a particular resource utilization by observing actions, a, and resources, r, signals, and comparing the shapes and signals characteristics, exploiting their understanding of the relation between action type and resources consumption. For example when it is found that a controllable action a_c, or an action dependent on some tuning activities, utilizes a resource close to saturation level, an action termination, or an execution redirection, can be applied to limit the excessive computation. Finding and understanding these relations is a key skill for any administrator controlling such a process.

However, in practice, actions use many resources and the underlying relation is often hidden, even from the expert. Uncovering the nature of the correspondence between systems interactions, utilization dependencies and performance characteristics requires longer monitoring, while at the same time any time–series of metrics gathered should be processed in a high–dimensional space in order to make decision about particular actions. This is related to various kinds of effects present in the complex environment enterprise systems operate, which are difficult to be anticipated fully before deployment. All these factors make the situation more challenging for human operators and administrators, especially when the problem dimensionality increases and the end–user requirements evolve.

In this paper we focus on a particular stage of the ASM process, modeling of the run–time dependencies between systems (Keller and Kar, 2000), with the aim to support analytics and decision support tools. To this end, we propose a method for decomposing ASM time–series signals, searching for hidden relations between users or systems activity and resources utilizations, controlling the impact of unexpected workloads – this is an area significantly underexplored.

In previous work we focused on the control problem, when no model of the enterprise system is available (Sikora and Magoulas, 2013). In (Sikora and Magoulas, 2014a), we reported on the use of methods that exploit signals similarity in order to establish causal dependencies. To this end, the so–called SCIC/SDCIC method was proposed (Sikora and Magoulas, 2014a). However, this approach proved to be sensitive to highly utilized resources– these are normally responsible for serving many actions executions requests.

The method proposed in this paper, named ASM-SD (Application Service Management Signals Deconvolution), alleviates this situation when decomposing mixed signals, so that more precise signals dependency and analysis of causal chains of events can be provided. The approach is based on decomposing ASM metrics time–series, outlining interdependencies between actions and utilized resources, and can be used to support human operators and administrators, or incorporated into decision blocks of autonomous controllers. In our methodology, ASM signals decomposition is formulated as a search problem that is tackled using evolutionary algorithms.

The paper is organized as follows. Section 2 describes the fundamentals of the problem area. Section 3 presents signals decomposition and details of the method proposed for the stated problem. Section 4 illustrates aspects of the performance of the approach under a few scenarios. Section 5 presents an outline of the signals decomposition optimization methods. Section 6 discusses details related to the validation of the global search methods tested. Section 7 completes the paper by outlining future work and providing concluding remarks.

2 Problem Description and Relevance

In an ASM system states can be defined by activity of services triggered by users or exposed interfaces, called actions, a, and corresponding resources, r, affected. Passive monitoring of API elements, threads, and sources of calls (e.g. other APIs, user actions or triggered events, authorization tokens etc.) allows identifying actions. Typically resources utilization is established through active monitoring (Sikora and Magoulas, 2013). Resources are most often related to elements of the hardware and software infrastructure (e.g. processors, memory, network or discs utilizations) but they can also define more abstract elements requiring synthetic calculations, e.g. SLAs operating as a quality measure for contracted services being provisioned, energy consumed, currently utilized infrastructure prices (present in SaaS or Cloud computing) and other performance indicators (Neugebauer and McAuley, 2001; Beloglazov and Buyya, 2010; Beloglazov et al, 2012), including revenue level or other business–oriented metrics.

2.1 ASM Systems Defined by Time–series of Metrics

Measurements define system responses and identify run–time characteristics effectively creating a model of the system performance characteristics. Considering the high–dimensionality of enterprise systems (Grinshpan, 2012) and the large size of the time–series databases, such a model is difficult to use. Moreover, there are many complex execution interdependencies in an enterprise system (Sydor, 2010; Sikora and Magoulas, 2013) that are effectively recognized while running to operate requested services.

Although a number of concurrently executing actions may have different resources usage characteristics (more details on this issue are provided in Section 4), effectively every action execution time is a result of resources, r, consumption and of time spent on each of the elements of the infrastructure. Thus the i–th action execution signal[1] is a function of resources utilized $a_i(t) = \alpha(r, a)$, $a_i(t) \in \{0, 1\}$ that depends on available system resources r and on other incoming or present actions executions a, forming a non-linear system.

Assuming that signals received from resources are convoluted with many action signals[2], then metrics representing actions execution, a, can be collated with time spent by the system on using resources, r, monitored (cf. with Equation 3 below). In other words, in this paper we search weights of strength (coefficients) of observed mixtures of actions, executed as resources are utilized, that allow the decomposition or deconvolution of observed signals. In contrast to approaches like standard Blind Signal Separation (BSS) and Semi-BSS (Bell and Sejnowski, 1995; Cichocki et al, 2002) in the ASM problem we know precisely, most of the time, shapes of the source signals that are formed from actions execution metrics; this is discussed further in Section 3.1. When a given resource is highly or fully utilized, actions using it are queued by the system. During saturation the signals are distorted/deformed. In such situations resources signals are flattened and extended in time, while actions execution time signals form much higher peaks. Thus the main assumption of most BSS methods is not met in the ASM context.

Actions, a, and resources, r, time–series define the system load–functional performance characteristics, delineating the input and outputs changes, and forming a $(\overline{\overline{a}} + \overline{\overline{r}})$–dimensional curve that is a trace of all system states.

In order to address the core problem that is to identify which software elements were responsible for specific resources utilization, in this paper, we propose a deconvolution technique to recover the original input signals. The method enables unmixing actions time taken from many monitored areas of a system and establishing coefficients m indicating how executing actions impacts resources usage.

2.2 Industry Practice

ASM typical practice is to calculate a time–series of action execution times to "action execution load". Such a transformation shows the count of actions running concurrently in time. This pre–processing approach often gives much better correlation, see Fig. 1. In process monitoring, this is a common way of aggregating changing runtime characteristics for better visibility and lower storage footprint of a number of parts of the system under control.

A critical operating region of system performance is reached when a resource is close to saturation (the whole scope of the resource is consumed). When this state is reached, the resource begins acting as a bottleneck for the rest of the system. Often though, this

[1] A signal as a time-series defines continuing flow of information, measured metrics or any other quantity that changes in time.

[2] Threads sleeps and resources waits form interesting situations. Although both sleeps and waits impact directly action execution times (an action does nothing as defined in the code) and active thread count, their relation to other resources utilization is very different. Whilst waits for resources are included in the resources utilization and queue lengths metrics, sleeps have minimal effect on CPU, Disk or other physical resources utilization.

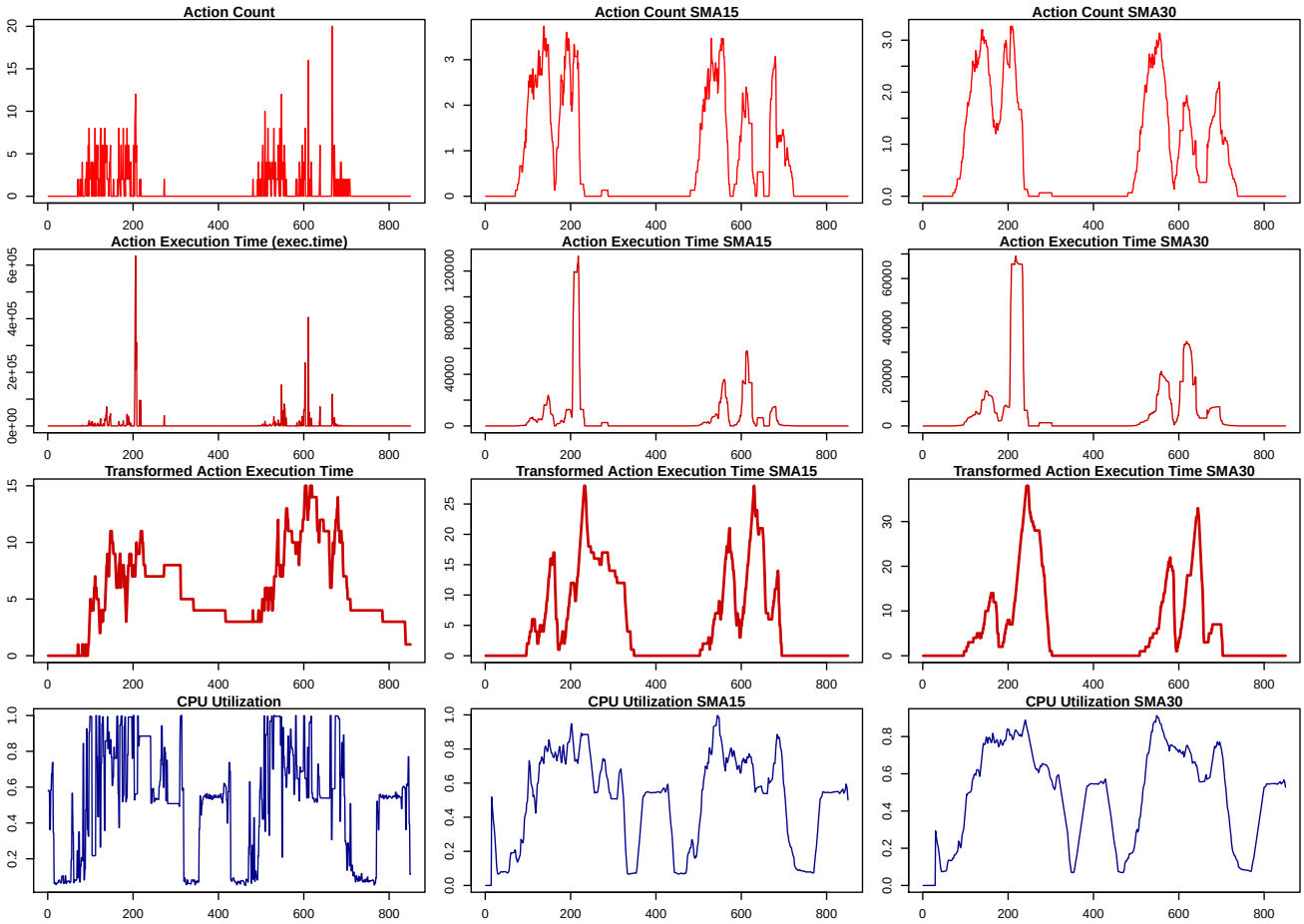


Fig. 1: Comparison of a sample time–series of CPU-bound action count and execution time in 1 second aggregates. Actions time–series are shown in red whilst resources utilized (CPU) time-series are in blue. The left column shows the time-series of the raw data- no smoothing was used. The middle column contains data processed with Simple Moving Average (SMA) that used a 15 secs wide aggregation window. The right column shows a SMA that used a 30 secs wide window. Rows starting from the top show: action counts (light red), action execution time (dark red), "action execution load" as transformed from execution time (dark red), and finally CPU utilization. It is worth noting the similarity between "action execution load" and CPU; this situation is further discussed in Section 3.1. These time-series data have been collected during a real– not model–based –experiment.

is actually the point when energy and amortized hardware are used optimally. There are of course concerns about stability and many other key system operations that have an impact on SLAs (Beloglazov and Buyya, 2010; Beloglazov et al, 2012), as well as concerns about the level of QoS and performance related issues, such as those related to queuing and batches processing.

Our approach treats the system as a black– or grey–box searching for relations between actions performance and observed use of resources. Although in most operating systems it is possible to track CPU or IO consumption time per process, or even thread, nevertheless, OS schedulers do not control the operation of applications internals. Hence, building such a model of relations at the application–level can greatly improve application and service–level controllers.

This is an area of particular relevance for other computation models as well, such as Cloud computing (Emeakaroha et al, 2011; Sironi et al, 2012; Yoo and Kim, 2013; Feng et al, 2012), Virtualization (Weng et al, 2011), and Map Reduce (Ibrahim et al, 2011), (Polo et al, 2011), where advanced resource and activity aware adaptive schedulers can be equipped with services activity and resources decomposition methods for better SLA and revenue awareness. An application level control and management is a widely adopted approach (Katchabaw et al, 1999; Chen et al, 2002), where control can be also done on a service level directly, with use of deeper

instrumentation of the application running in ASM environment.

In that respect, embedding the proposed ASM-SD method can potentially enhance the performance of adaptive schedulers, especially those working on the application level (Bartolini et al, 2012), or the scheduling policy, extending the predictive modeling of available green energy (Aksanli et al, 2012).

As it will be shown in Section 6, experiments provide evidence that the proposed ASM-SD method can be successfully implemented with use of global search algorithmic techniques.

3 Signals Decomposition as a Search Problem and the ASM-SD Methodology

The proposed Application Service Management Signals Deconvolution (ASM-SD) method tackles the problem from an optimization perspective. The approach was built on the assumption that the resource utilization at time point t is a result of actions metrics at this point:

$$r_k(t) = \sum_i m_{ik}(r_k)a_i(t) + n_k \ , \tag{1}$$

where r_k is the k-th resource, a_i is the i-th action, m_{ik} is an unknown coefficient of a component, or service, which causes changes in r_k after a_i calls, and n_k is noise that can be considered as an error. The presence of the error vector n_k is related to many factors, e.g. noise present in the system that may consist of unwelcome and often hidden components such as software sleeps, waits for unmeasured resources, other OS level or software level framework operations, or unmonitored system activity that utilizes system resources. It is normally advised to assume that this vector consists of non-negative values, as the above mentioned situations normally introduce additional execution times against the resources utilized, i.e. resources are used more often than monitored action times suggest.

Equations of this form are common in the BSS class of problems (Bell and Sejnowski, 1995; Cichocki et al, 2002) mentioned earlier in Section 2.1, and can be reformulated as a system of linear equations of the following form:

$$R = MA + U \tag{2}$$

where R, A are vectors of resources and action time–series respectively, $R = [r_1(t), ...]^T, A = [a_1(t), ...]^T$, matrix M contains mixing coefficients, and U is the residual vector.

That formulation would suggest applying Linear Models Fitting (LMF) (Chambers, 1992; Wilkinson and Rogers, 1973; web, 2014c) to solve this problem, but the LMF approach tackles the optimization problem in a symmetric way, where the equation may be equally under– or over– determined, and it is difficult to adjust it considering all additional calculations that are specific to the ASM area, e.g. boundary limits that are *asymmetric* due to causality direction, and penalties for rare execution or low total execution time, see Section 3.1. Thus, more details on the optimization function implementation are needed and, effectively, more flexible optimization methods have to be tested.

As mentioned above, in the proposed approach the unmixing process is formed as a search problem that minimizes a single-objective. Considering that there are many nonlinearities, which have an impact on action execution times a and resources utilized r, as outlined in Section 2, the optimization of $f(x) : \mathbb{R}^n \to \mathbb{R}$ can be considered as a general Non-linear Optimization Problem (NLP) (Ruszczyński, 2006), defined as follows:

$$\arg\min \left(r_k(t) - \sum_i \Big(m_{ik}(r_k)a_i(t) \Big) \right) = n_k \ , \tag{3}$$

where m_{ik} denotes unknown entries of the mixing matrix M of size $\overline{\overline{r}} \times \overline{\overline{a}}$, for which the equation in brackets attains its minimal value. M is transformed to a solution point s in the search space $S \subset \mathbb{R}_+^{\overline{\overline{a}} \cdot \overline{\overline{r}}}$.

There are, however, several constraints involved: (i) signals are by definition non-negative $a \geq 0, r \geq 0$ and normalized, and (ii) potential negative values of n_k are subject to additional penalty in the optimization function implementation (discussed further in Section 3.1), based on the assumption that actions form the inputs and resources are the system outputs, which are impacted by the results of the actions. Thus, a possible effect cannot be prior to the activity that generated it.

Assuming that there is an adequately rich set of activity measurements A gathered to explain the resources utilized R, n_k as a residual, or fitting error, of unobservable signals is minimal. However, if the solution point s, as a vector of specific values m_{ik} of the mixing matrix M, results in a substantially large residual error, then the k-th resource signal is not matching the observed activity, and therefore either : (i) the monitored activity does not adequately explain the resources usage (many important actions have not been monitored), or (ii) the resource usage does not reflect the activity signals gathered (e.g. this is often present in memory consumption cases).

Eq. 4, below, is an extension of Eq. 3 that factors in information about actions execution times span and

match against a resource, considering the time spent on the execution up to the current time instance t_1 when the rest of the collected samples are aggregated:

$$\arg\min \left(r_k(t) - \sum_i \left(m_{ik}(r_k) \int_0^{t_1} a_i(t)\,\mathrm{d}t \right) \right) = n_k \ . \quad (4)$$

Thus the approach of Eq. 3 can be used mainly for very short executions times, or where sampling time or aggregation is wider than most of actions executions, that is quite common in ASM databases. The approach of Eq. 4 can be used for much longer actions (e.g. batch jobs) that often have significant impact on systems performance and consist of different phases, with different resources usage footprint. Very long executing actions could be organized in sequences of monitored sub-actions considered under the assumptions of Eq. 3. Moreover, we assume that when action execution times are short, or the span of the time–series aggregate used is wider than most of actions executions, then Eq. 4 approximates Eq. 3.

In other words, the general approach presented above is based on similarity and shapes matching between different domains: activities and resources usage, where units are different. Thus, it should rather not consider values comparison directly. Therefore all signals should be normalized before any further processing is done, and the equation $R = MA + U$ is considered to be a good approximation of the system.

Still in situations where activity levels (execution counts or times) are far different between actions in A, the precision can be enhanced when such information is factored in as a weighted vector of action execution times. Actions less often used (thus having lower total execution times) should be considered as having less impact on resources and effectively their error should be augmented as part of another tier of penalty setting.

An iterative optimization process searches for suitable coefficients of the mixing matrix M. Solving the problem by a greedy approach to calculate the objective function for every designed dimension n of k–th values would be $O(n^k)$, such a brute–force search may be acceptable for low–dimensional cases only. As mentioned earlier the search space S dimensionality is a product of $\overline{\overline{A}} \cdot \overline{\overline{R}}$, where typically in a real system we can have tens to hundreds of resources and hundreds to thousands of actions.

Each iteration of the method focuses on all resources R. If an action is found during its execution to use time from many different resources, then for a particular action implementation the usage should be consistent over time. Consequently it has been assumed that m_{ik} is constant for the i–th action a and the k–th resource r at iteration t, see Eq. 3. Depending on the application context, this assumption may be subject to revision after significant code, parameters or input data changes, or when other factors, like intrusive load or control actions, change the normal system responses behavior. Although this situation is not considered in this paper, we can briefly comment that a way to potentially overcome this issue is to consider threating actions running with different parameters or input data, or after changes in the release of the software – this however needs careful consideration as it may lead the count of dimensions to explode, which in turn could limit the method used.

Lastly, it is worth mentioning the possibility of lowering the count of actions by aggregating actions signals with similar load–functional characteristics that give comparable result on the usage of resources. Then, signal unmixing analysis can be used to confirm any similar relations between actions of different types and resources impact.

3.1 The ASM-SD Methodology

The proposed methodology includes the following signal processing steps.

Step 1. Signals Normalization
: Time–series data are linearly scaled, so their amplitudes are standardized to the range of $[0, 1]$. This is required because in practice actions and various resources signals are expressed in different units, and values gathered are difficult to compare directly. Emphasis here is on the signals shapes under consideration rather than the specific values gathered and aims at evaluating all dimensions (e.g. resources, actions) equally.

Step 2. Signals Denoising
: System administrators are used to interpret scalar values that represent various system dimensions, like resources utilization and systems actions (aggregated values), and associate their changes in time. This is a simple but effective way of aggregating data and lowering the scale of resolution of the metrics time–series to be stored. Most of the monitoring tools actively gathering data about systems actions collect counts and execution times continuously through software components (collectors), which are weaved into application runtime (Grinshpan, 2012; Haines, 2006). Such raw metrics are stored in various internal storages for further use by metrics collection tools and need to be aggregated for presentation and further times–series processing purposes. Simple Moving Average (SMA) (Pyle, 1999) as a low

pass filter has been used[3]. This form of signals filtering allows enhancing the level of similarity between the input action and the resources signals which play the role of the system's response– see Fig. 1.

Step 3. Transformation of the Executions Load
Next, an "action execution load" transformation is executed for each action signal. The essence of the transformation is to change execution time measurements collected at the end of the action call[4] into a time–series that represents the count of actions being executed at a given point in time. Such a transformation performs signals smoothing, which is useful when signals contain many peaks, and helps enhancing the level of similarity further– see comparison presented in Fig. 1.

Fig. 2 illustrates all stages of the ASM-SD methodology. It includes an additional step of dimensionality reduction which filters out system actions that have lower impact. In ASM industry practice, the number of action signals A greatly exceeds the number of resources R. Thus reducing the actions dimensionality can improve the ASM-SD execution time. Actions impact can be measured by ranking selected action types– e.g. those which have higher cumulative execution times– or highly called actions[5], or, even in certain cases, by using a weighted product of those two ranking criteria.

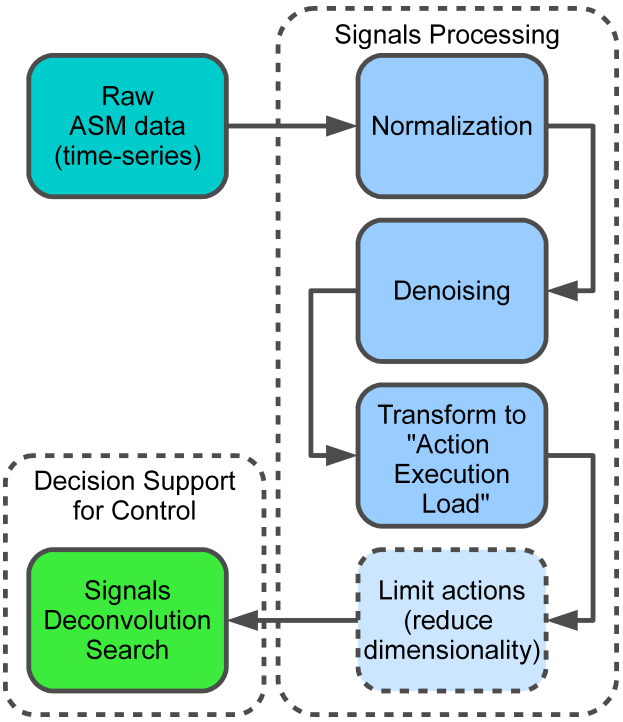


Fig. 2: ASM system incorporating the ASM-SD methodology.

3.2 The ASM Search Space

Seeking appropriate coefficients of the mixing matrix in Eq. 3 requires searching a multidimensional space. Below, a simple ASM system containing four actions and a single resource, where the intensity of the actions and usage of the resource are different per action type (see Fig. 3), is used as an example to illustrate features that are common to ASM systems. The search is conducted in a 4–dimensional normalized hypercube, and denoising of all signals is achieved through SMA filtering that follows a window size of 15 secs– see Step 2 in Section 3.1.

In Fig. 3, different slices of the search space are shown by projecting the search surface along two dimensions, which represent pairs of actions against a single utilized resource. This visualization could help gaining some insight about the characteristics of the optimization problem. Potential minimizers in these 3D and 2D plots are marked with a blue diamond. Obviously, these are only slices and the full search space explored by the ASM-SD method has higher dimensionality, so the coordinates of the actual problem minimizers differ from the ones shown in these figures. In fact, the ASM-SD search space appears to be multimodal. This is also illustrated in the 2D contour plots, which show that there are neighborhoods of local minima (see the plots at the upper right side of the diagonal in Fig. 3) surrounded by the contour lines. The space is complex with irregular gradient values, from very narrow barrier related slopes (discussed later), to flat regions with many widespread local minima, and valleys– see the 3D plots at the lower left side of the diagonal in Fig. 3. From our observations in the experiments reported next, the region surrounding a good minimizer is flat and often wide. Moreover, often the neighborhood of the best solution found contains many shallow local minima that are not very different in terms of function values to the "global" minimum.

As mentioned earlier, in Section 3, negative values of residuals of Eq. 3 should be avoided, and to this end *bouncing boundaries* are used in the optimization function implementation. This is done through the introduction of penalties, whose values are passed to the

[3] As a rule of thumb a fair level of denoising is achieved when SMA aggregates values following the width of the longest action.

[4] In engineering practice in most of the cases action executions are assigned at the time the action started.

[5] The execution count can very often cause software or hardware resources utilization regardless of the execution time.

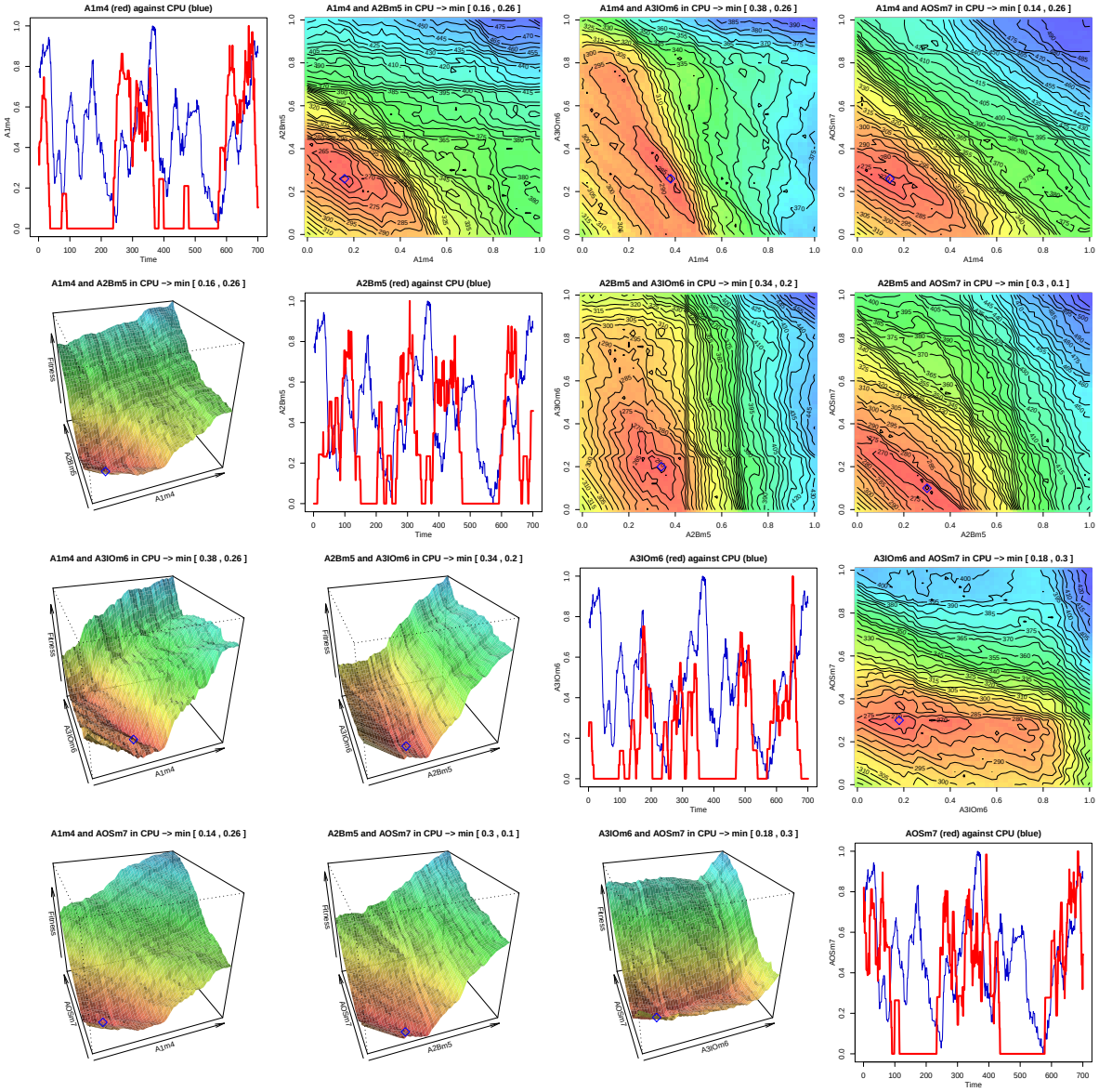


Fig. 3: The search space of an ASM system having four actions and a single resource being utilized. The figure illustrates the search space with respect to each pair of actions.The original action signals are shown in the plots placed along the main diagonal. The 3D surfaces and the corresponding 2D contour plots depict the fitness function with respect to a pair of isolated action types. The 3D surface plots (at the lower left side of the diagonal) provide a good view of the shape of the fitness function along two dimensions, whilst the corresponding contour plots (at the upper right side of the diagonal) provide a better view of the various local extrema (areas of closed isolines). The lowest point in each of these projections indicates a minimizer and has been marked with a blue diamond.

methods as a parameter, that are added to the error for every negative coefficient and residual value (Wright and Nocedal, 1999). Such an approach forms barriers for matching signals whose total impact exceeds resource utilization. Any candidate solution that violates the problem's assumptions/constraints is removed from the feasible region by assigning a penalty to it.

4 Testbed Design and Experiments

In this paper we are focusing on evolutionary methods to solve the ASM–SD problem, as this approach appears to be eminently suitable due to the characteristics of the search space, discussed in Section 3.

The test–bed framework of Fig. 4 has been implemented to allow us to explore the potential use of the

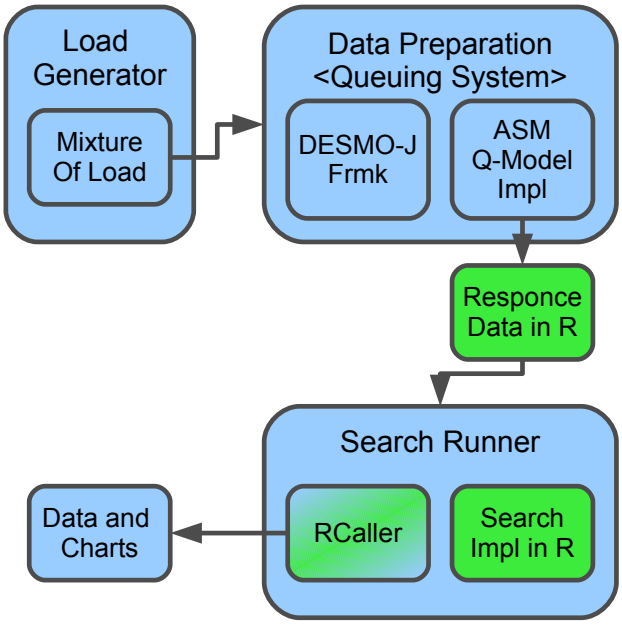


Fig. 4: The ASM system test framework.

proposed approach in the industry practice. It provides an ASM environment where iterative experimentation can be conducted, collecting performance data for the various methods tested and generating a rich set of ASM data for testing population–based methods under various loads several times.

Previous research in the ASM field showed that the minimal time for a load test – rich enough in terms of different load pattern changes – is around 1 to 2 hours long (Sikora and Magoulas, 2013). To alleviate the problem of conducting long simulation runs queuing model simulations were used. This is a well known simulation modeling approach, as discrete event simulation modeling in the form of queuing systems has a long history of applications (Lazowska et al, 1984; Banks et al, 2000; Harchol-Balter, 2013).

To this end, we reviewed several existing event–based frameworks for systems modeling and workload analysis, such as GridSim toolkit (Buyya and Murshed, 2002), GroudSim (Ostermann et al, 2011), Java Modeling Tools (Bertoli et al, 2009, 2006), Palladio (Becker et al, 2009) and CloudSim (Buyya et al, 2009; Calheiros et al, 2011). We found that the framework based on DESMO-J (Page et al, 2005; Gehlsen and Page, 2001; web, 2014b) is more appropriate due to simplicity, insight and flexibility of direct elements instrumentation for metrics gathering that are required in our case.

DESMO-J utilizes plain Java code for the model and the queue networks framework, which allows to integrate it with the rest of the proposed framework[6].

The software framework provides APIs for forming custom topologies of queue networks, utilizing many different actions, adding custom counters for SLAs and putting customized load in a flexible manner. It is a pure–old Java objects approach (POJO), so it is ready to be equipped with a controller actuators to simulate environments under service control to test operation and performance of the computer system during the design phase, or be applied as a model base for a weaved-in controller to be instrumented directly in the application code to manage and regulate the application behavior in operation.

Fig. 4 presents the design of the test–bed that consists of the: *Load Generator* that defines execution parameters for the modeled system; Data Preparation block that provides the enterprise system model that generates ASM metrics, which are required to prepare a dataset for the *Search Runner* that uses various evolutionary computing methods.

All empirical evaluations were done with the use of `R` scripting, which can be directly called from a Java–hosted ASM controller. The authors used the approach of RCaller that was proposed and described in Satman (2014). Experiments utilized RCaller v2.1.1 implemented by Satman (2013) as an `R` integration library for Java applications.

4.1 The Load Generator

The *Load Generator* is a software component that is responsible for creating input data for the ASM model deployed in the Data Preparation system– see Fig. 4.

Identifying bottlenecks, overused or saturated components/resources, and monitoring the entire performance of the modeled system are key functional prerequisites of the test–bed design in order to cover operational areas in our experiments that are interesting from systems performance perspective. Moreover, the model can be a subject of runs under various load conditions in order to geta an insight of the system dynamics. The *Load Generator* provides an API that gives the necessary flexibility of defining the load mixture by specifying arrival and execution rates and distributions of each of the action types individually. Action type definition contains a vector of load sources that specifies how the arrival rates are distributed in time during simulation.

[6] After applying a simulation–based data generator, the speed increased by a factor of approximately 600 to 800 (that practically allows running 24hrs load simulation in about 2 to 3 minutes, depending on the count of the load sources).

Each action must be also equipped with measures showing proportions of available resources utilization.

Such an approach gives a very concise yet flexible way of modeling a load coming to the system from many sources (users interface, web services and scheduled batch jobs) using the system concurrently. The test–bed can execute simulation from very simple signals mixture to very complex, multi-action workloads.

The following example provides a concrete scenario where actions play the role of input to the queue model. Normally to test the ASM deconvolution process of Eq. 3 under a mixture of signals, a set of many concurrently executing actions of many types needs to be run.

The example is based on a single action type definition that is rather CPU–bound, but also utilizes Disk in only 5% of the execution time, with the following execution distribution parameters: exponential distribution with given arrival rate $Exp(\lambda = 10)$; action execution time defined with use of normal distribution $\mathcal{N}(\mu = 5, \sigma^2 = 0.1)$; a variable load defined as a call probability pattern given by the sequence $p = [0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.1, 0.0, 0.0, 0.1, 0.4, 0.4, 0.6, 0.2, 0.1, 0.0, 0.0]$ repeated 3 times in the time of a single experiment[7]. For better comparison, the corresponding Java code snippet is provided below:

```
new ActionType("A3IO2",
  new LoadPattern(        // arrival
    new LoadDistExponential(
          "DistrExp", 10),
      new double[]{       // intensity
        0.0, 0.2, 0.2, 0.2, 0.2, 0.2,
        0.1, 0.0, 0.0, 0.1, 0.4, 0.4,
        0.6, 0.2, 0.1, 0.0, 0.0},
        3),
  new LoadDistNormal(     // execution
        "DistrNorm", 1, 0.1),
  new ResourcesUsage[]{ // resources use
    new ResourcesUsage(
          ProcessingType.CPU, 0.95),
    new ResourcesUsage(
          ProcessingType.Disk, 0.05)},
  new SLA());
```

[7] The load pattern gives a precise way for configuring variability in the expected intensity of the frequency of incoming calls for a given action. This parameter is very helpful to define special test cases, such as actions interference using the same resource, receiving temporarily higher load to observe effects of spikes in resources consumption, or short reoccurring load changes to analyze any "delay" impacting the strength of the ASM-SD.

4.2 Model and ASM Data Preparation

The computer system model implementation in Fig. 5 applies Processor Time Sharing approach, where each job is sliced to atomic subtasks that, as per action type characteristics (functionality and load), have different resources distributed (Lazowska et al, 1984; Harchol-Balter, 2013).

As mentioned earlier, DESMO-J (Page et al, 2005) has been found suitable for this study. More details on available frameworks and comparisons are provided in Göbel et al (2013). For the benefit of this study the framework provides implementations of CPU and IO load simulations, based on those resources queues where there is no waiting line and all jobs receive an equal proportion of the service capacity. That is essential in order to model a simple operating system dispatcher which is a building block of every computing system. Based on our experience the artificial data generated by the model of this queuing system design match the metrics collected during real load runs. Rerunning the simulation is reproducing hours of load test in seconds.

It is worth mentioning that more advanced computer systems, including distributed components (network delays), multi-core (lower load queues as per simple processor sharing) or farms of servers covered with load-balancers (each of them are multiprocessor with a disk controller) are feasible to be implemented in the framework used. Nevertheless the extensions are not needed in order to illustrate the operation of the ASM-SD method – as in engineering practice most of the activity and resources metrics can be isolated as per machine or sub-system, and consequently the problem is reduced to a single server again. Of course adding resources (network delays, wait to connection pools etc.) to the picture would increase the dimensionality of the search space– see Section 3.2.

4.3 The Search Runner

As mentioned earlier in Section 3.2, due to the multiple minima and high–dimensional nature of the ASM-SD search space, the experimental study focused on comparison of different population–based methods.

Several series of experiments were performed on synthetic datasets to confirm the effectiveness of the approach. Three different population–based search strategies available in `R` are presented in this paper: PSO (Clerc

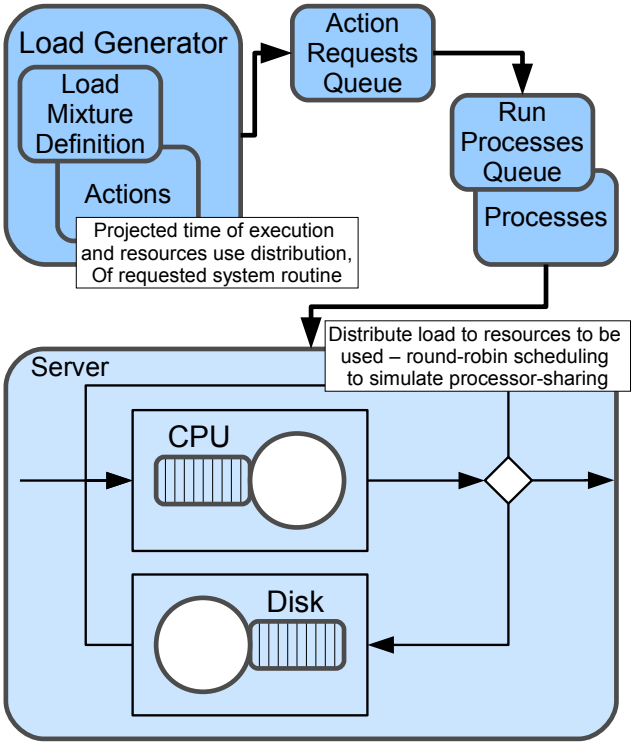


Fig. 5: *Load Generator* and ASM system model for signals generation.

et al, 2011)[8], GAoptim (Tenorio and Tenorio, 2013) [9], and DEoptim (Ardia et al, 2011; Mullen et al, 2011) [10].

The *Search Runner* component provides a software framework abstracting access to different APIs of the search methods `R` implementations. As `R` optimization implementations, based on the approach taken by `optim {stats}`, PSO, GAoptim, and DEoptim differ in input parameters, it was necessary to wrap the access with a single access layer giving input parameters abstraction.

In order to compare the speed of convergence, precision, robustness, and general performance of the different methods we use as our main performance criteria the execution time and the residual error– see Eq. 3. More detailed description of the evolutionary methods is provided next in Section 5.

[8] A Particle Swarm Optimization (PSO) implementation consistent with the standard PSO 2007/2011 of Maurice Clerc et al. (Bendtsen, 2012), version 1.0.3 (2012-09-02).

[9] Genetic Algorithm (GA) optimization package for real–based and permutation–based problems, version 1.1 (2013-03-24).

[10] The DEoptim package implements the Differential Evolution (DE) algorithm for global optimization of a real–valued function of a real–valued parameter vector, version 2.2-2 (2014-12-17).

Interested readers can find a performance comparison of non–population–based, general–purpose optimization methods in (Sikora and Magoulas, 2014b), where Simplex (Nelder and Mead, 1965) and Simulated Annealing (SANN) (Bélisle, 1992) were tested. Preliminary experiments in Sikora and Magoulas (2014b) found that population–based methods find good ASM-SD solutions earlier, and are generally resilient to the problem of local minima.

Fig. 6a presents an example response and unmixed action signals using the proposed approach, where one of the actions signals is a result of the Java code snippet presented in Section 4.1.

5 Search Methods Performance and Parameters Tuning

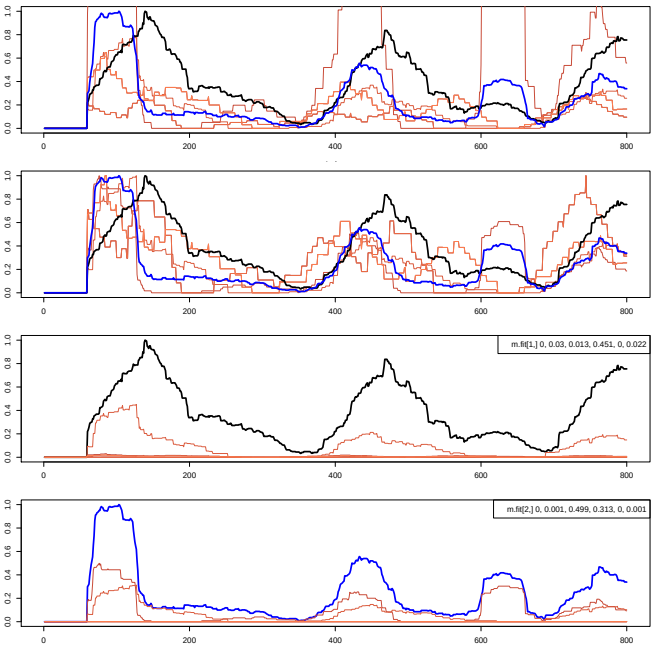
This study is not tackling meta-optimization, where another overlaid optimizer for parameter calibration (tuning) is being used (Grefenstette, 1986; Pedersen, 2010a,b). The purpose of this work is to explore the different methods and validate their applicability. Thus, much work has been done on visualization comparison of the optimization characteristics from various perspectives.

This section provides insights in the examined search methods performance when tested under different experimental conditions and tuning modes. Series of experiments have been executed checking different load situations (variable arrival rates and difference load patters).

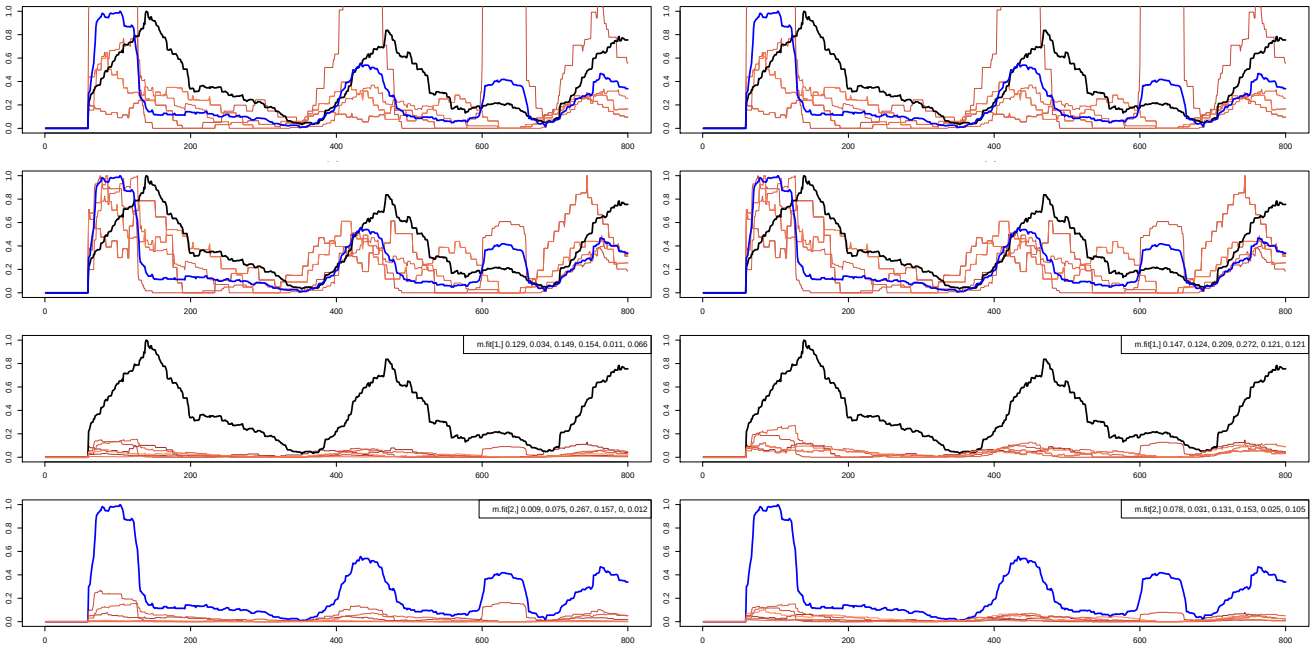
The experiments focused on `R` implementations only, giving a good comparison ground for selected population–based optimization. Typical for this class of global search methods metrics like maximum number of iterations, population size and others have been gathered.

Due to the large number of individual simulation runs conducted, the visualization of multivariate datasets focused on entire distributions rather than statistically aggregated values for groups of data. Thus the methods comparisons in Figs. 7– 16, use violin plots (Hintze and Nelson, 1998) that better shows individual experiments than box–and–whiskers plots (McGill et al, 1978). To highlight the directions of the changes Local Polynomial Regression Fitting (loess) trend line is used (Cleveland, 1979; Cleveland and Devlin, 1988; Cleveland et al, 1992). To generate the figures `R {graphics}` and ggplot2 (Wickham, 2009) plotting systems were used.

The following sections present the experimental results.



(a) PSO



(b) DE (c) GA

Fig. 6: Comparison of deconvolution process for three optimization methods: PSO, DE and GA. The time series were captured by the *Search Runner* debugging during one of the runs, where six different concurrently executing actions are present. Most of the actions are mainly CPU–bound but also use Disk with different proportions of CPU and Disk usage, one of the actions is rather Disk–bound. Resources are denoted by black and blue thick lines, whilst actions are shown as thin lines using different tones of red. Each of the groups shows from top to bottom: original action values signals, all normalized, unmixed actions impacting CPU and Disk respectively.

5.1 Particle Swarm Variants

In this section, variants of Particle Swarm Optimization are presented. They use a population of simple agents interacting locally to develop collective behavior in a decentralized fashion, in order to find the best solution in the search space (Clerc, 2010).

Particle Swarm Optimization (Clerc and Kennedy, 2002; Clerc et al, 2011) implementation is consistent with the standard PSO2007/2011 of Maurice Clerc et al. (Bendtsen, 2012), version 1.0.3 (2012-09-02).

Figure 7 shows a comparison of the two variants. The default swarm sizes (population size) for the two methods differ: SPSO2007 defaults to $\lfloor 10+2*\sqrt{\overline{\overline{A}} * \overline{\overline{R}}} \rfloor$, while SPSO2011 use 40. This has been considered in the figures showing comparison as a function of population size that is always relative (in %) to the default value.

Comparison of SPSO2007 and SPSO2011 has been done for default 1000, 700, 500 and 300 max iterations (maxit) in eight–dimensional space deconvolution problem– see Fig. 7. Results confirm that performance of SPSO2007 and SPSO2011 is very similar. The experiment shows that even quite low swarm sizes, e.g. 10–15 swarm size, give good solutions, with low error and short execution times.

5.2 Genetic Algorithms

This group of global search methods is based on a population of candidate solutions that iteratively evolve toward better solution points (in terms of fitness value) in the search space. Each candidate solution, often called individual, is defined as a set of genotype data, which provide a basis for selection, crossover, and mutation operations that take different forms depending on the method. The population of individuals in each iteration is called a generation (Michalewicz, 1996).

The Genetic Algorithm of `GAReal` {`GAoptim`}, version 1.1 (published 2013-03-24) (Tenorio and Tenorio, 2013) has been tested.

Due to complex implementation of custom crossover, selection and mutation operators default functions have been used. Thus, the `GAReal` function for real–based optimization has been tested under the following conditions. *Selection*: the default option performs a fitness proportionate selection, so that the fittest individuals will have greater chances of being selected. *Crossover*: this used the blend option that performs a linear combination of the individuals chromosomes, and so introducing new information into the resulting offspring, with the crossover rate equal to 0.9 (see Figs. 8–9) as probability of two individuals effectively exchanging genotype. *Mutation*: the implementation uniformly samples given mutation rate, (default 0.01) multiplied by population and present dimensions, mutation points along the population matrix, each sampled locus is replaced by a random-uniform number between 0 and 1. *Elite rate*: it uses a default value of 0.4, as ratio of the best–fitted individuals amongst the whole population that are automatically selected for the next generation.

As it will be discussed later (cf. with Fig.14), the GA has been found to be very fast (in terms of execution time) in high dimensional cases, over 50 dimensions.

5.3 Differential Evolution Variants

Differential Evolution (DE) introduced by Storn and Price (1997) has been applied to problems from a variety of domains. ASM-SD experiments used `DEoptim` {`DEoptim`}, version 2.2-2 (published 2014-12-17) (web, 2014a; Ardia et al, 2011; Mullen et al, 2011).

The choice of DE parameters, population members count, NP, crossover probability, CR, and differential weighting factor, F, can have an impact on optimization performance (Storn and Price, 1997). Therefore selecting the DE parameters that provide a good performance has been thoroughly researched. Nevertheless, our experiments – results summarized in Figs. 10–16 – show that in the ASM-SD problem changes in default parameters have limited influence on the method performance and therefore default values are suggested.

A comparison of six DE strategies available (web, 2014a; Ardia et al, 2011; Mullen et al, 2011) is presented in Fig. 10, crossover probability values are shown in Fig. 8 and population members count is exhibited in Fig. 16. DEoptim performance as function of maximum set iterations (maxit) is shown in Figs. 10, 11 and 12.

DE finds solution points with much lower error values than GA, with very similar execution times, in low–dimensional cases (Fig. 8). In high–dimensional cases, execution times of DE are much higher than those of GA (Fig. 9).

This method has been found to possess the longest execution times especially in highly–dimensional cases, but generated quite low Error values (Fig. 14). DE is the most sensible method for low dimensions count, Fig. 15.

All DE strategies applied to ASM-SD problem give very similar results. All of them give very similar error and execution time characteristics. Fig. 11 and 12 show comparison of all six strategies under different load, maximum iteration and dimensions involved conditions.

The best combination for a general ASM-SD use of differential weighting F and crossover CR was found to be: F = 0.1 and CR = 0.7 (Fig. 13).

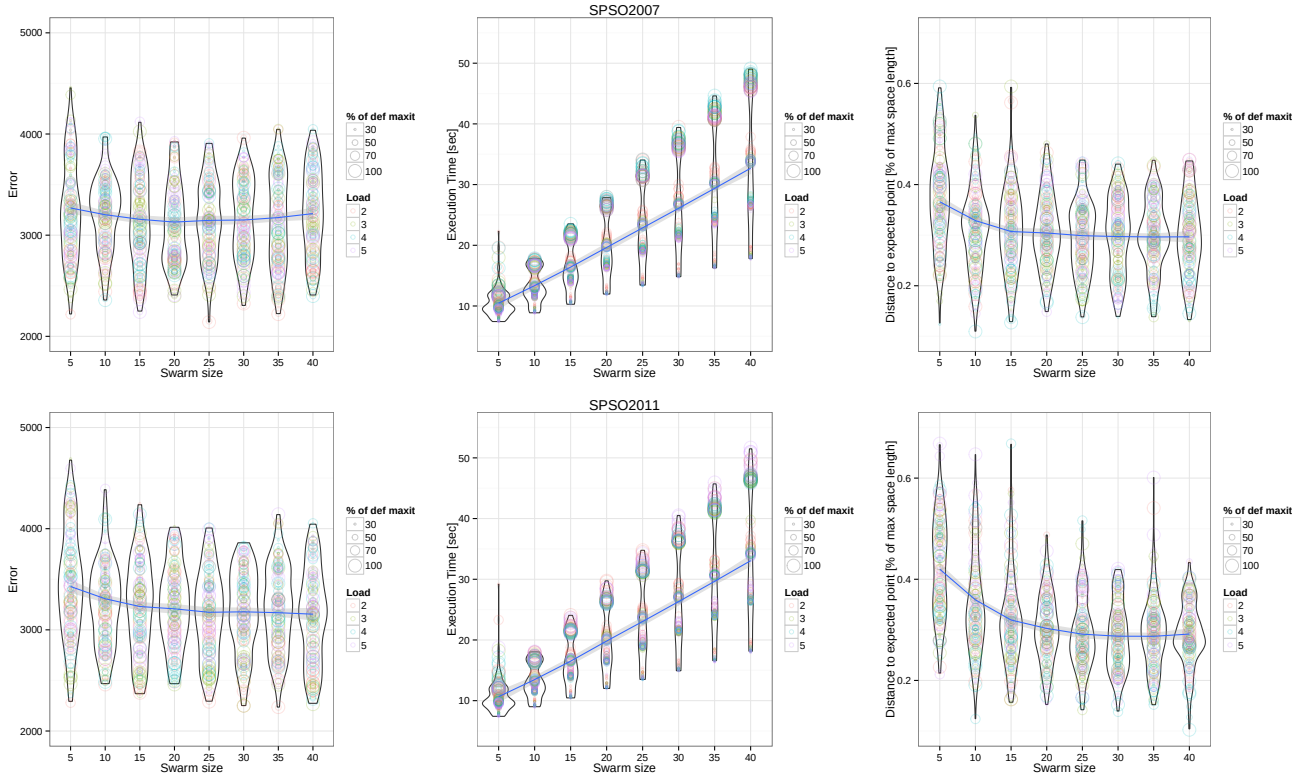


Fig. 7: Comparison of SPSO2007 and SPSO2011 variants of PSO.

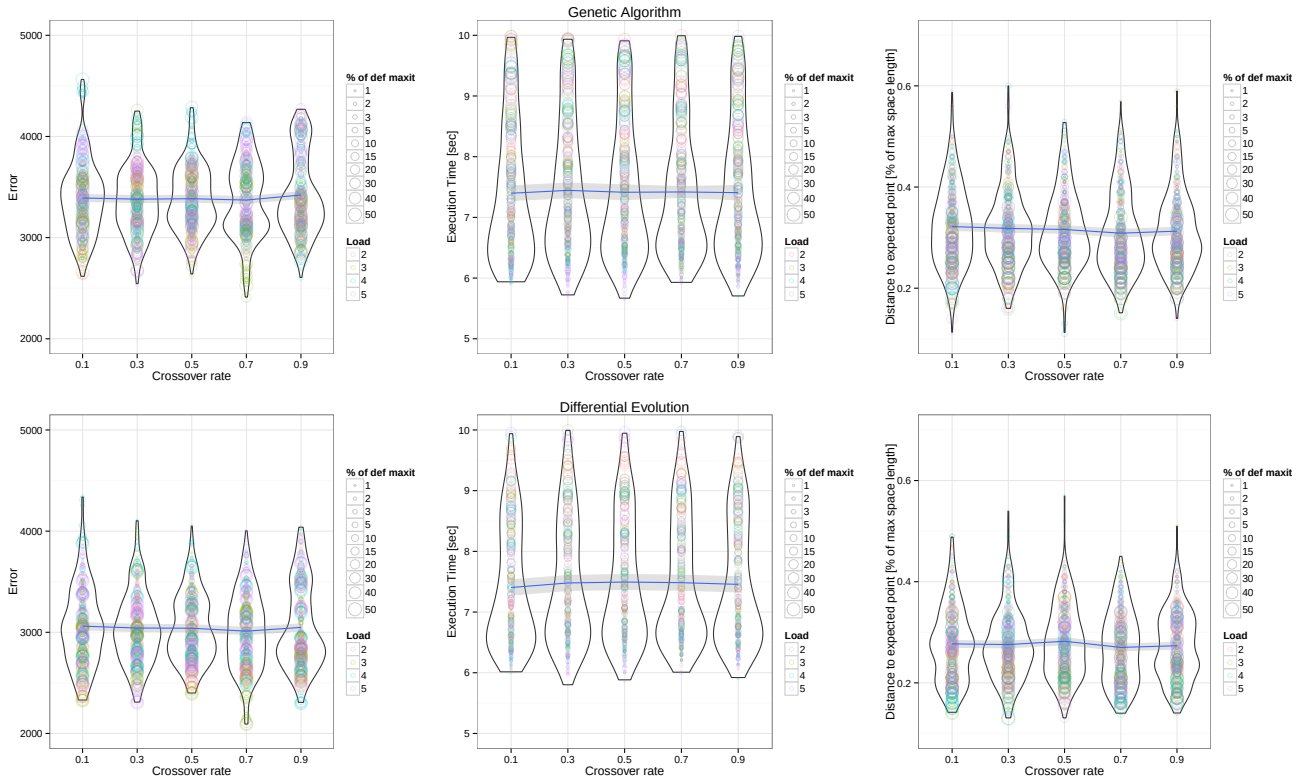


Fig. 8: Comparison of Crossover parameter of GA and DE using low–dimensional ASM-SD problem set.

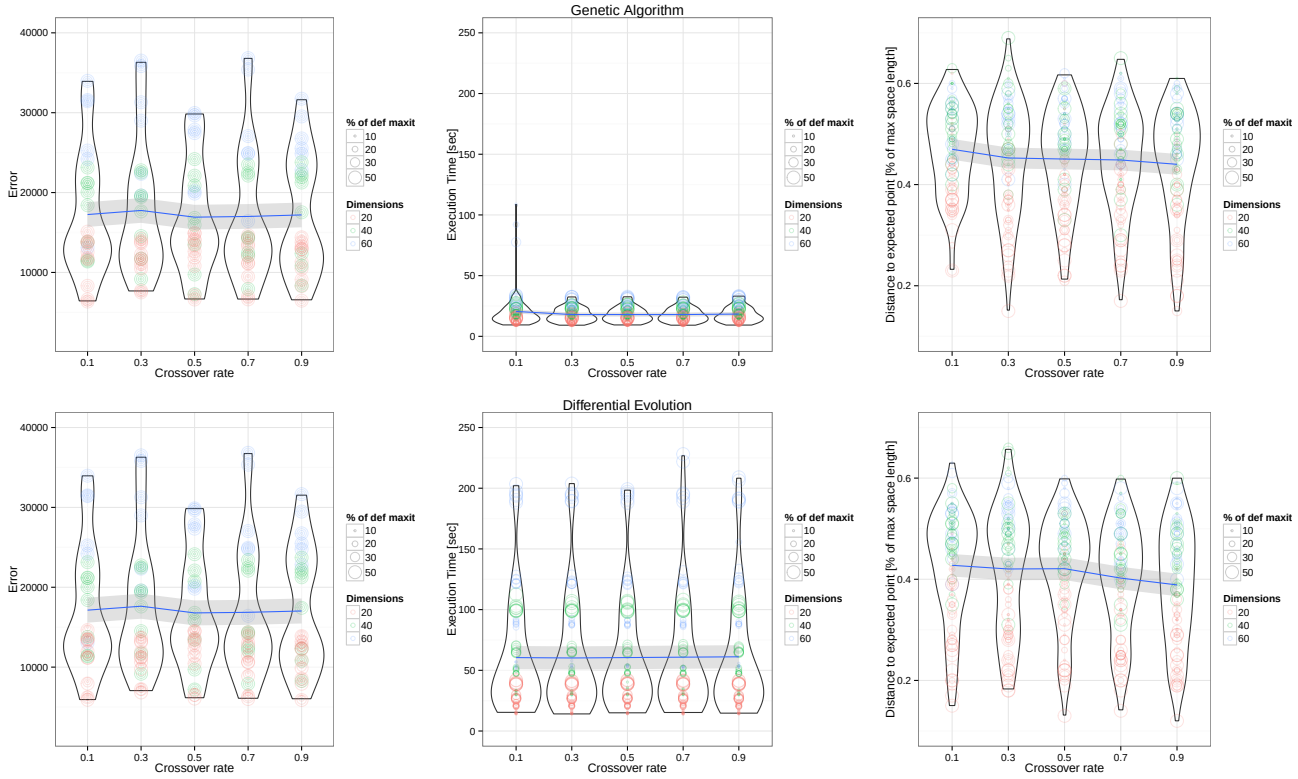


Fig. 9: Comparison of Crossover parameter of GA and DE using high–dimensional ASM-SD problem set.

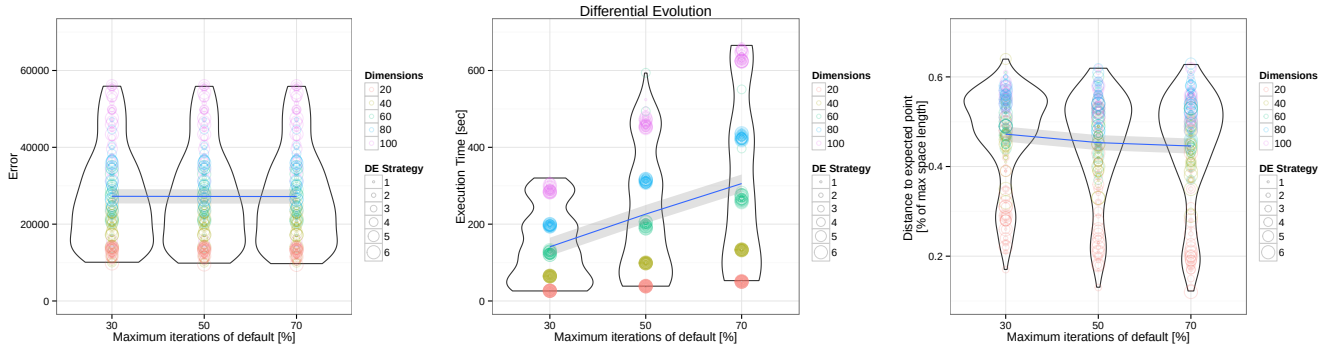


Fig. 10: Differential Evolution performance as function of % of default maximum iterations (maxit) parameter.

6 Search Methods Quality Validation

The main selection criteria in the ASM context for selecting a particular search method is performance, which typically is established based on search cost (as per Eq. 3), execution time and the distance of the solution point from the expected result point (precise for simpler, low–dimensional cases, where modeled system is not overloaded) – this is given as % of maximum space length (that is the search space diagonal of the n–dimensional normalized hypercube, equal to $\sqrt{n}$).

This section presents the above measures for each method with respect to load, dimensions (a function of actions and resources), and maximum iterations.

6.1 Execution Runs

All experiments have been carried on a single CPU machine (all runs on the same platform, R + JVM + OS + hardware) in order to test a serial processing execution times in a rigorous fashion.

The first set of experiments focused on signals decomposition (ASM-SD) of the model containing 10 di-

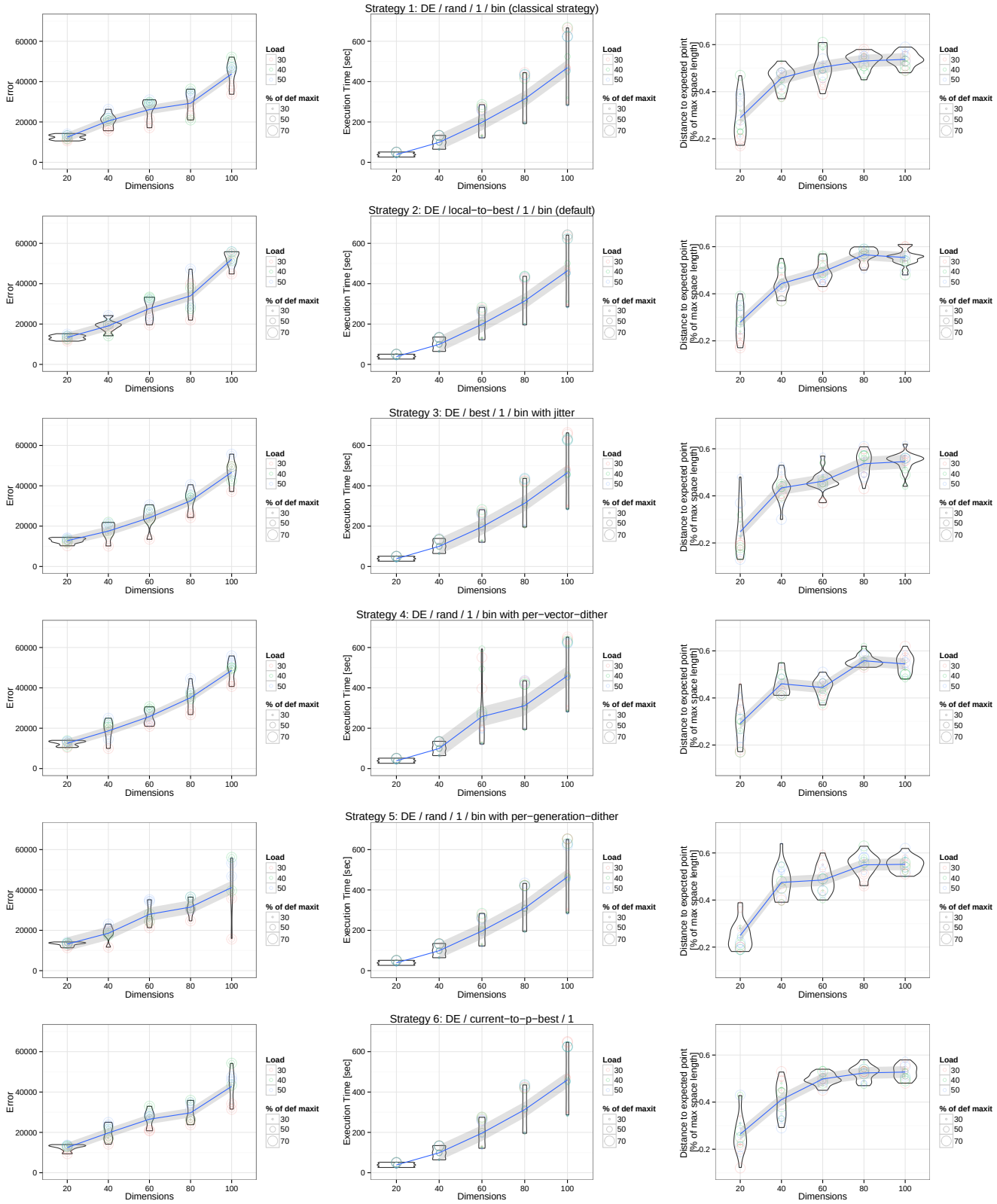


Fig. 11: Comparison of Differential Evolution strategies performance with respect to the problem dimensions.

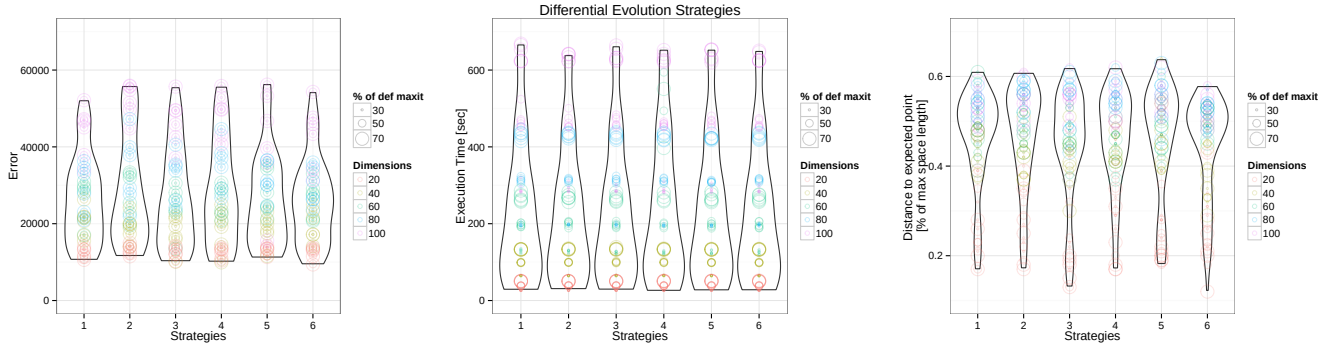


Fig. 12: Comparison of Differential Evolution strategies under different dimensions and maximum iterations conditions.

mensions, five action types running with use of two resources. To explore the behavior of the methods further a series of experiments were conducted using a model containing between 10 to 100 dimensions, and variable arrival rates and load patterns (Fig. 14, 15, 16).

In order to validate the method, we must establish an error measure such as the residual of Eq. 3. Since *Load Generator* parameters are under strict control (see Section 4.1), it is possible to derive analytically an expected solution point containing values of actions impact on resources, using load model parameters, i.e. arrivals, execution time distribution and load distribution in time.

The results of the experiments show that typically: (*a*) execution time is exponentially dependent on maximum iterations (maxit), Fig. 10, (*b*) execution time is linearly dependent on populations used, Fig. 16. In contrast, error characteristics remain flat with respect to either population or max iterations. Thus it would make sense to limit maximum number of iterations (maxit) and population.

Due to the ASM-SD search space nature, e.g. flat regions around optimum (as discussed in Section 3.2), neither the maximum number of iterations (maxit) nor the count of population set for a given search run change the error substantially. Similar error rates are found even for very small maxit values, indicating that each of the methods tested was able to find low error values very quickly (see the first column on Fig. 15). However, as the search operates further, reaching higher maxit, it finds points of slightly smaller errors. Small differences in error values are produced by different search space points that have a significant impact on the distance to the expected solution point. This is shown in the third column of maxit– and population– based comparisons in Fig. 15 and Fig. 16 respectively.

The main factor impacting error is the number of dimensions and consequently the size of the search space, as shown in Fig. 14. The search (execution) time, shown in the middle column of the figures, is primary impacted by maxit and the population size used. Generally the fastest method is GA. SPSO2007 is slightly slower but often it converges closer to the expected point (Fig. 15 and Fig. 16).

6.2 Industrial Applicability

Experiments show that the ASM-DS method gives stable results very early; for example within 20–30% default maxit (see Fig. 14 and 15). Thus, in engineering applications limiting the maximum iterations to 20% could balance the good performance/value trade-off.

Also the impact of population size can be limited, as shown in Fig. 16 where the error characteristics as function of the population count remain almost constant.

In the real world context, a system with a 100 leading action types and a minute long time aggregate, collected over a single release cycle, which is usually one month long, generates a dataset that contains approximately 43200 samples. The processing time is around 40–50 minutes, under the run–time conditions used in the experiments (e.g. considering the average GA execution time for a 100–dimensional system of 800–samples long dataset), which can be easily implemented in a scheduled task of an ASM framework.

7 Conclusions and Future Work

The ASM area still lacks research in data analysis methodologies to support ASM operators and improve performance of autonomous controllers.

This paper presented an approach that can be applied as a deconvolution technique to help uncovering hidden run–time relations between observed signals in

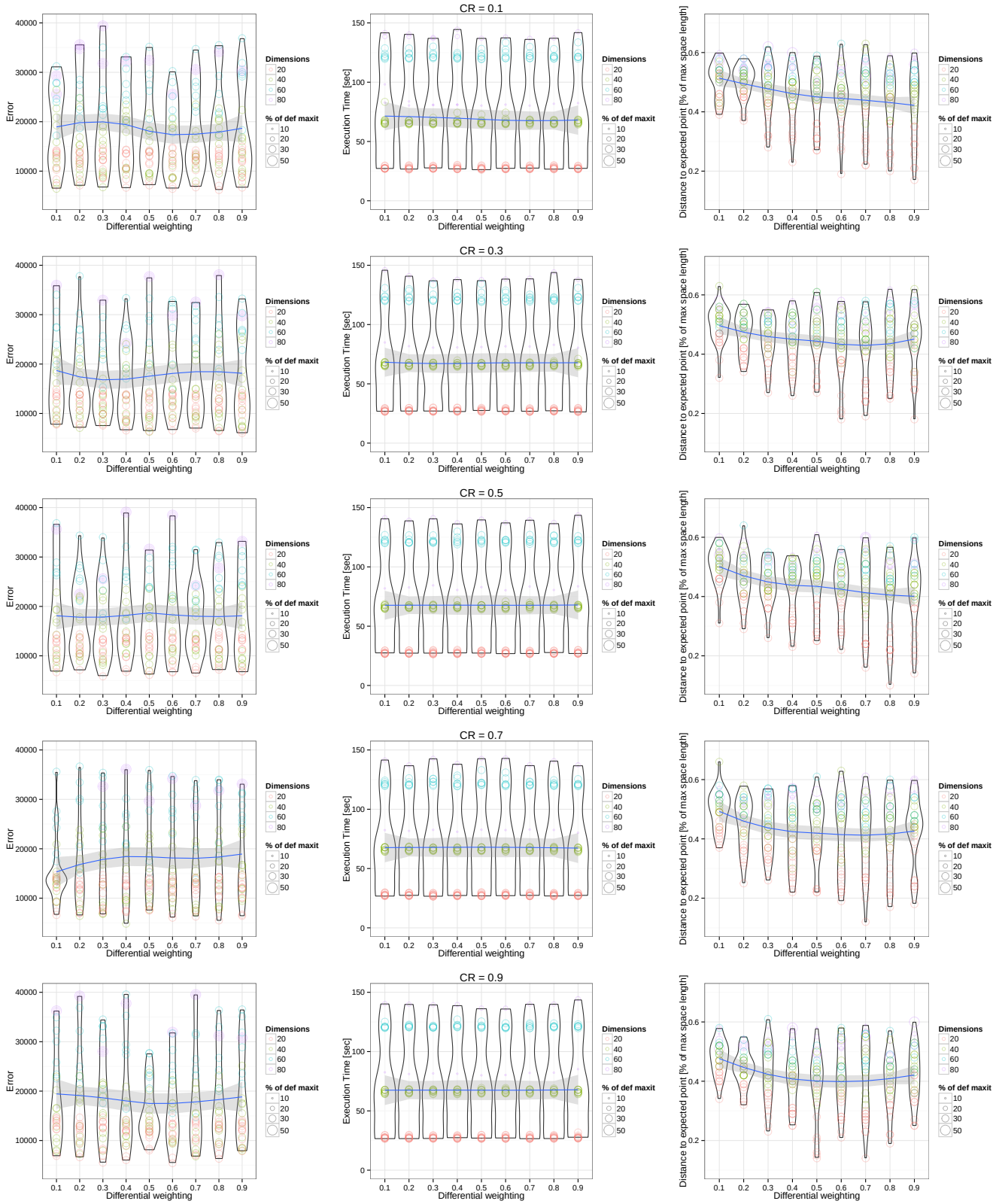


Fig. 13: Comparison of Differential Evolution (strategy 2 – default) under different crossover probability CR and differential weighting factor F conditions.

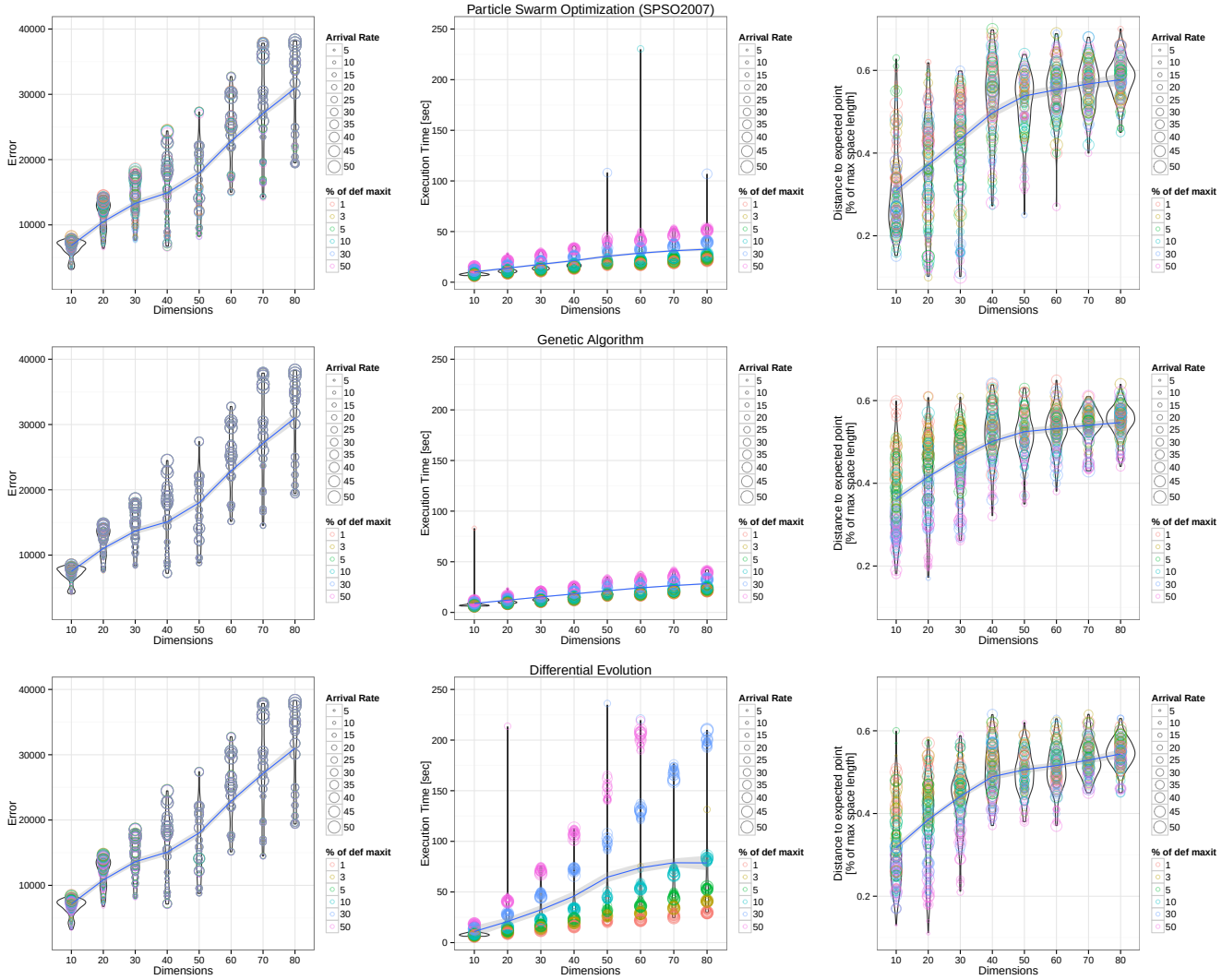


Fig. 14: Comparison of error, execution time and distance to expected solution point with respect to dimensions count.

the ASM field. Metrics signals gathered during normal operation of the enterprise system build a database of time–series data that effectively contains definitions of hidden causality chains present in the system.

In order to extract the causal relations and establish models of systems interactions, utilization dependencies and performance characteristics that a controller or administrator can exploit for further decision making, matching of running application responses under given conditions with resources usage is required.

The high–dimensionality of the search spaces, the size of the multivariate datasets and the morphological characteristics of the search space require special qualities from the specific search methods applied.

To this end, in this work we described a signal deconvolution method driven by evolutionary search that can be applied in ASM data analytics and adaptive controllers, and investigated the potential of Particle Swarm, Genetic Algorithms and Differential Evolution methods in this context.

Our tests confirmed the practical potential of the population-based type of search with GAs being the fastest method amongst the tested ones. PSO is very well performing, is close to GAs in terms of execution time, and the solution points found are often closer to an expected optimal point.

Parallelization of the populations, even though supported by some of the tested methods (DE, PSO), has not been included in this study but it was left for future work. Also scalability issues require further investigation, especially as there is much interest in the community for parallel population based metaheuristics (Ned-

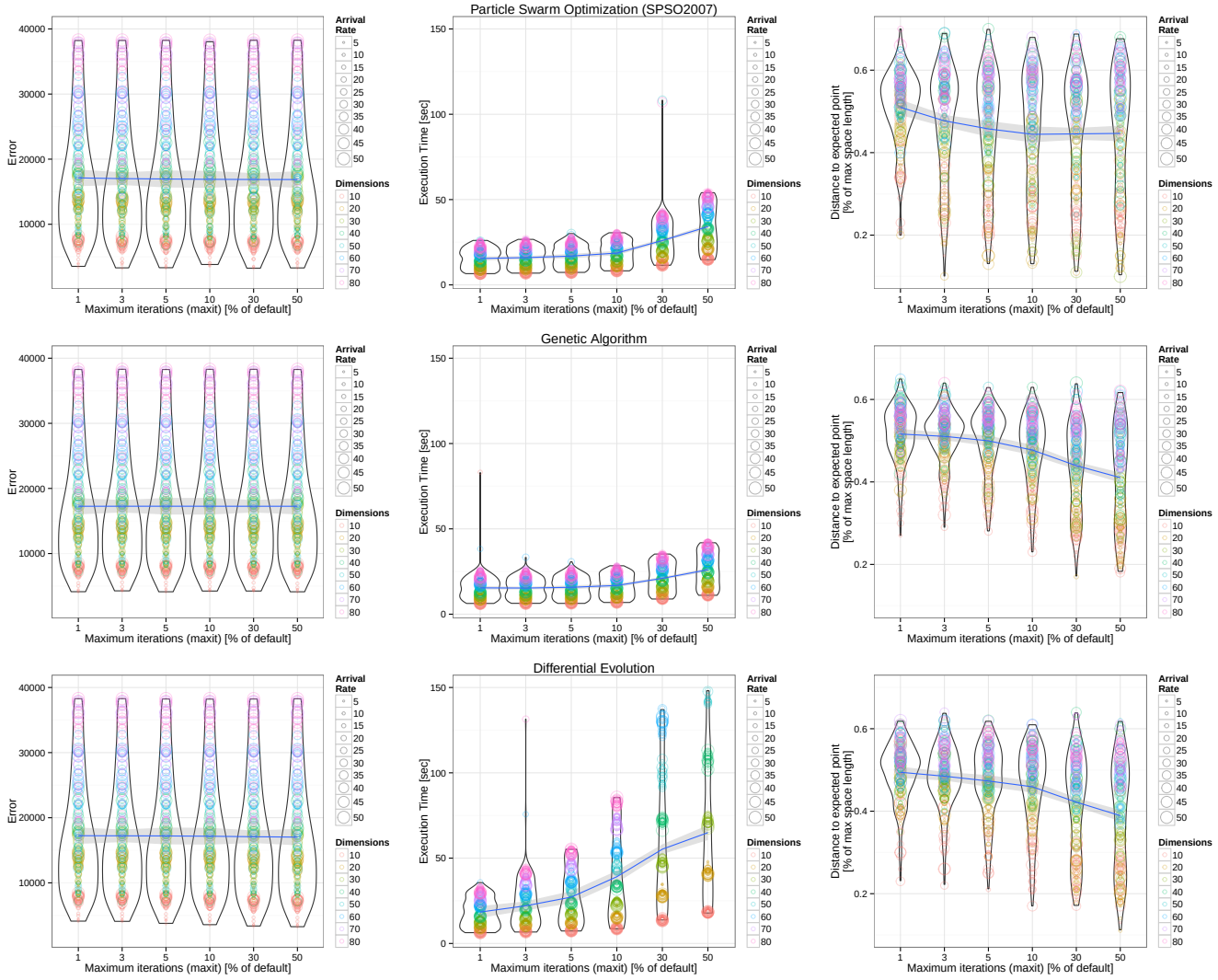


Fig. 15: Comparison of error, execution time and distance to expected solution point as function of maximum iterations (maxit).

jah et al, 2006), where concurrent search runs interact to improve the overall solution. Furthermore, our future work investigates ways to extend the deconvolution approach, enhancing the proposed method to better tackle issues related to specific types of resources and actions signals, system responses delays, signals noise, impacts of unmonitored blind spots, and observer effects, to better address needs of adaptive controllers, schedulers, but also decision support systems. Moreover, additional research needs to be done in processing signals that are acquired from more complex computing system models, containing many architectural units, components, network elements and communicating with other distributed services to support the integration needs of modern computing systems. All these issues are linked to our ongoing work on both autonomous and human-driven control in ASM environments that is equipped with soft computing and machine learning methods.

Acknowledgements This work was partially supported by Solid Software Solutions[11].

References

(2014a) DEoptim: Global optimization by differential evolution. URL `http://cran.r-project.org/web/packages/DEoptim/`

(2014b) DESMO-J: A framework for discrete-event modeling and simulation. URL `http://desmoj.sourceforge.net/`

[11] `http://www.solidsoftware.pl/`

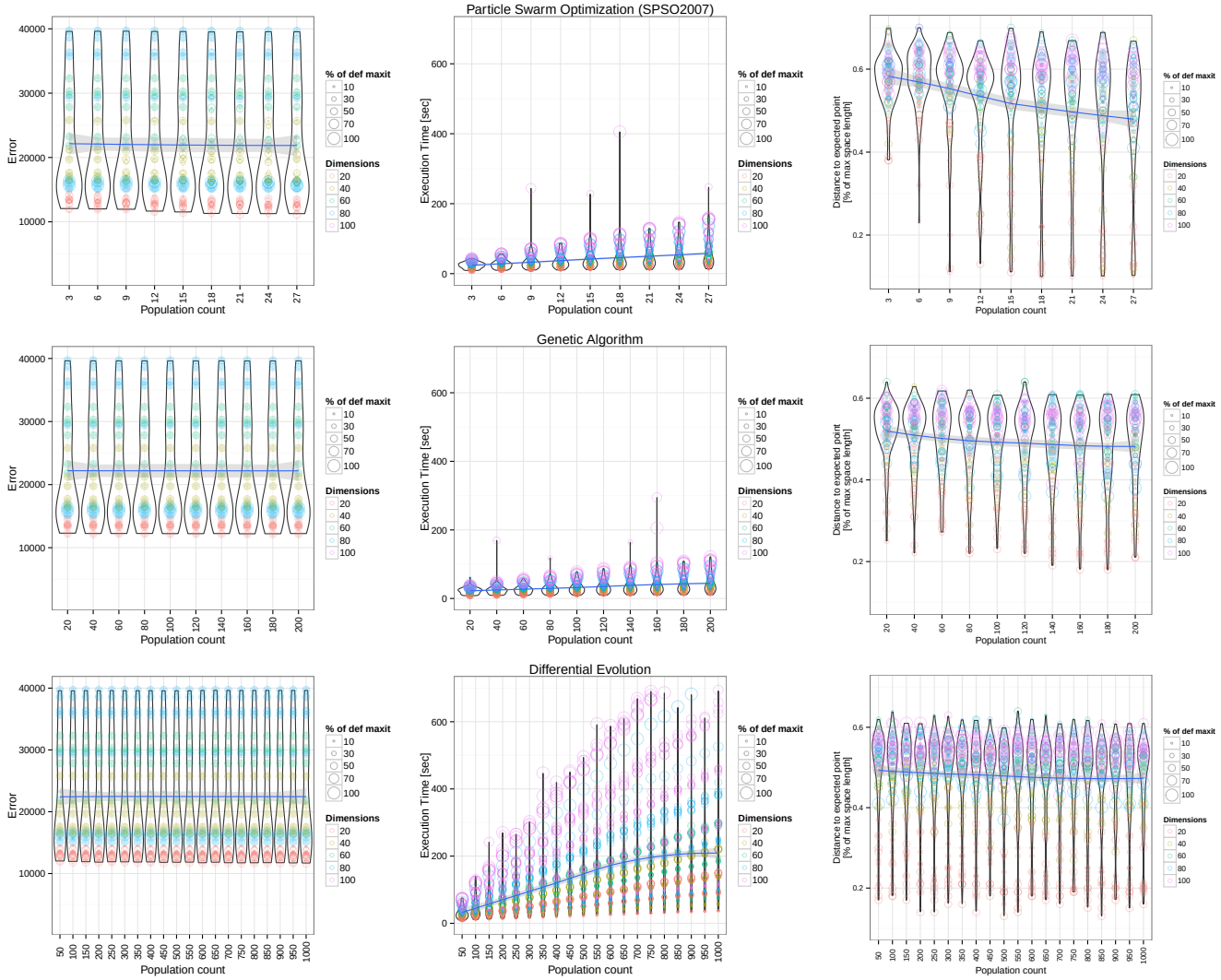


Fig. 16: Comparison of error, execution time and distance to expected solution point with respect to population count.

(2014c) Fitting linear models, lm stats, R Documentation. URL `http://stat.ethz.ch/R-manual/R-patched/library/stats/html/lm.html`

Aksanli B, Venkatesh J, Zhang L, Rosing T (2012) Utilizing green energy prediction to schedule mixed batch and service jobs in data centers. ACM SIGOPS Operating Systems Review 45(3):53–57

Ardia D, Boudt K, Carl P, Mullen KM, Peterson BG (2011) Differential evolution with DEoptim. The R Journal 3(1):27–34

Banks J, Carson J, Nelson B (2000) DM Nicol, Discrete-Event System Simulation. Prentice Hall

Bartolini DB, Sironi F, Sciuto D, Santambrogio MD (2012) An infrastructure to instrument applications and measure performance in self-adaptive computing. In: Workshop on Self-Awareness in Reconfigurable Computing Systems (SRCS), Citeseer, p 44

Becker S, Koziolek H, Reussner R (2009) The Palladio component model for model-driven performance prediction. Journal of Systems and Software 82(1):3–22

Bélisle CJ (1992) Convergence theorems for a class of simulated annealing algorithms on R^d. Journal of Applied Probability pp 885–895

Bell AJ, Sejnowski TJ (1995) An information-maximization approach to blind separation and blind deconvolution. Neural computation 7(6):1129–1159

Beloglazov A, Buyya R (2010) Energy efficient resource management in virtualized cloud data centers. In: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE Computer Society, pp 826–831

Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Computer Systems 28(5):755–768

Bendtsen C (2012) PSO: Particle Swarm Optimization, {pso}, R Documentation. URL `http://cran.r-project.org/web/packages/pso/`

Bertoli M, Casale G, Serazzri G (2006) Java modelling tools: an open source suite for queueing network modelling andworkload analysis. In: Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on, IEEE, pp 119–120

Bertoli M, Casale G, Serazzi G (2009) JMT: performance engineering tools for system modeling. ACM SIGMETRICS Performance Evaluation Review 36(4):10–15

Buyya R, Murshed M (2002) GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and computation: practice and experience 14(13-15):1175–1220

Buyya R, Ranjan R, Calheiros RN (2009) Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In: High Performance Computing & Simulation, 2009. HPCS'09. International Conference on, IEEE, pp 1–11

Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience 41(1):23–50

Chambers C (1992) The design and implementation of the self compiler, an optimizing compiler for object-oriented programming languages. PhD thesis, Stanford University

Chen MY, Kiciman E, Fratkin E, Fox A, Brewer E (2002) Pinpoint: Problem determination in large, dynamic internet services. In: Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on, IEEE, pp 595–604

Cichocki A, Amari Si, et al (2002) Adaptive blind signal and image processing. John Wiley Chichester

Clerc M (2010) Particle swarm optimization, vol 93. John Wiley & Sons

Clerc M, Kennedy J (2002) The particle swarm-explosion, stability, and convergence in a multidimensional complex space. Evolutionary Computation, IEEE Transactions on 6(1):58–73

Clerc M, et al (2011) Standard PSO 2011. Tech. rep., Technical Report [online] http://www. particleswarm. info/, Particle Swarm Central

Cleveland W, Devlin S (1988) Locally weighted regression: an approach to regression analysis by local fitting. Journal of the American Statistical Association 83(403):596–610

Cleveland WS (1979) Robust locally weighted regression and smoothing scatterplots. Journal of the American Statistical Association 74(368):829–836

Cleveland WS, Grosse E, Shyu WM (1992) Local regression models. Statistical models in S pp 309–376

Emeakaroha VC, Brandic I, Maurer M, Breskovic I (2011) Sla-aware application deployment and resource allocation in clouds. In: Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual, IEEE, pp 298–303

Feng G, Garg S, Buyya R, Li W (2012) Revenue maximization using adaptive resource provisioning in cloud computing environments. In: Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing, IEEE Computer Society, pp 192–200

Gehlsen B, Page B (2001) A framework for distributed simulation optimization. In: Proceedings of the 33nd conference on Winter simulation, IEEE Computer Society, pp 508–514

Göbel J, Joschko P, Koors A, Page B (2013) The discrete event simulation framework DESMO-J: Review, comparison to other frameworks and latest development. In: ECMS, pp 100–109

Grefenstette JJ (1986) Optimization of control parameters for genetic algorithms. Systems, Man and Cybernetics, IEEE Transactions on 16(1):122–128

Grinshpan L (2012) Workload characterization and transaction profiling. Solving Enterprise Applications Performance Puzzles: Queuing Models to the Rescue pp 57–94

Haines S (2006) Pro Java EE 5 performance management and optimization. Apress

Harchol-Balter M (2013) Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge University Press

Hintze JL, Nelson RD (1998) Violin plots: a box plot-density trace synergism. The American Statistician 52(2):181–184

Ibrahim S, Jin H, Lu L, He B, Wu S (2011) Adaptive disk i/o scheduling for mapreduce in virtualized environment. In: Parallel Processing (ICPP), 2011 International Conference on, IEEE, pp 335–344

Katchabaw MJ, Howard SL, Lutfiyya HL, Marshall AD, Bauer MA (1999) Making distributed applications manageable through instrumentation. Journal of Systems and Software 45(2):81–97

Keller A, Kar G (2000) Dynamic dependencies in application service management. IBM Research Report

RC 21770 (97933)
Lazowska ED, Zahorjan J, Graham GS, Sevcik KC (1984) Quantitative system performance: computer system analysis using queueing network models. Prentice-Hall, Inc.
McGill R, Tukey JW, Larsen WA (1978) Variations of box plots. The American Statistician 32(1):12–16
Michalewicz Z (1996) Genetic algorithms+ data structures= evolution programs. Springer
Mullen KM, Ardia D, Gil DL, Windover D, Cline J (2011) DEoptim: An R package for global optimization by differential evolution. Journal of Statistical Software 40
Nedjah N, Alba E, de Macedo Mourelle L (2006) Parallel Evolutionary Computations. Springer
Nelder JA, Mead R (1965) A simplex method for function minimization. The Computer Journal 7(4):308–313
Neugebauer R, McAuley D (2001) Energy is just another resource: Energy accounting and energy pricing in the Nemesis OS. In: Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on, IEEE, pp 67–72
Ostermann S, Plankensteiner K, Prodan R, Fahringer T (2011) GroudSim: An event-based simulation framework for computational grids and clouds. In: Euro-Par 2010 Parallel Processing Workshops, Springer, pp 305–313
Page B, Kreutzer W, Gehlsen B (2005) The Java simulation handbook: simulating discrete event systems with UML and Java. Shaker Verlag
Pedersen MEH (2010a) Good parameters for particle swarm optimization. Hvass Lab, Copenhagen, Denmark, Tech Rep HL1001
Pedersen MEH (2010b) Tuning & simplifying heuristical optimization. PhD thesis, University of Southampton
Polo J, Castillo C, Carrera D, Becerra Y, Whalley I, Steinder M, Torres J, Ayguadé E (2011) Resource-aware adaptive scheduling for mapreduce clusters. In: Middleware 2011, Springer, pp 187–207
Pyle D (1999) Data preparation for data mining, vol 1. Morgan Kaufmann
Ruszczyński AP (2006) Nonlinear optimization, vol 13. Princeton University Press
Satman MH (2013) RCaller: A library for calling R from Java URL `https://code.google.com/p/rcaller/`
Satman MH (2014) RCaller: A software library for calling R from Java. British Journal of Mathematics & Computer Science 4:2188–2196
Sikora TD, Magoulas GD (2013) Neural adaptive control in application service management environment. Evolving Systems 4(4) pp 267–287
Sikora TD, Magoulas GD (2014a) Finding relevant dimensions in application service management control. In: Intelligent Systems for Science and Information, Springer, pp 335–353
Sikora TD, Magoulas GD (2014b) Search-guided activity signals extraction in application service management control. In: Computational Intelligence (UKCI), 2014 14th UK Workshop on, IEEE, pp 1–8
Sironi F, Bartolini DB, Campanoni S, Cancare F, Hoffmann H, Sciuto D, Santambrogio MD (2012) Metronome: operating system level performance management via self-adaptive computing. In: Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE, IEEE, pp 856–865
Storn R, Price K (1997) Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization 11(4):341–359
Sydor MJ (2010) APM Best Practices: Realizing Application Performance Management. Apress
Tenorio F, Tenorio MF (2013) Package gaoptim URL `http://cran.r-project.org/web/packages/gaoptim/`
Weng C, Liu Q, Yu L, Li M (2011) Dynamic adaptive scheduling for virtual machines. In: Proceedings of the 20th international symposium on High performance distributed computing, ACM, pp 239–250
Wickham H (2009) ggplot2: elegant graphics for data analysis. Springer
Wilkinson G, Rogers C (1973) Symbolic description of factorial models for analysis of variance. Applied Statistics pp 392–399
Wright S, Nocedal J (1999) Numerical optimization, vol 2. Springer New York
Yoo S, Kim S (2013) Sla-aware adaptive provisioning method for hybrid workload application on cloud computing platform. In: Proceedings of the International MultiConference of Engineers and Computer Scientists, vol 1